GloServ: Global Service Discovery Architecture

Knarig Arabshian and Henning Schulzrinne
Department of Computer Science
Columbia University, New York, NY
{knarig,hgs}@cs.columbia.edu

Abstract

Due to the growth in ubiquitous computing technology in the past few years, the need for context-aware service discovery across wide area networks is becoming prevalent. We propose GloServ, which is a global service discovery architecture that locates services throughout wide and local area networks. It supports services encompassing different domains such as events, people or places. Services can be described semantically using the Resource Description Framework (RDF) and can be queried using the RDF Query Language (RQL). GloServ hierarchically defines services using RDF schemas and assigns each service a URI according to its location within the hierarchy. The hierarchical architecture for GloServ is similar to how domain names are categorized in DNS. Service discovery can either be initiated by the user or by the system. For automated service discovery, users are detected with sensors and are presented with services available according to their preferences. Graphical user interfaces for querying data is dynamically generated through the processing of RDF data.

1 Introduction

GloServ is a global service discovery architecture that addresses the need for wide area service discovery for ubiquitous and pervasive computing. It operates on wide area as well as local area networks. Any type of service is handled which may include events, location-based services, communication or web services.

There are many different scenarios global service discovery systems are useful in. Imagine an avid traveller who is interested in different events or services that are taking place in a particular city. A New Yorker visiting Paris, for instance, wants to be notified of all the concerts taking place in Paris. When he lands in the airport, his device will issue a query pertaining to his preference, such as classical music concerts, and then try to locate classical concerts in that area. Another example would be of a doctor visiting another hospital who wants to discover different medical services. The doctor issues a query and obtains the types of services she is interested in automatically.

A few characteristics of GloServ distinguish it from other service discovery systems. It can apply to different service contexts or networking environments because services can be defined and described in a flexible hierarchical ordering using the Resource Description Framework (RDF) [5] [14]. The hierarchical architecture for GloServ is also similar to how domain names are categorized in DNS [17]. This results in efficient administration and querying of services as well as scalability. Additionally, GloServ provides dynamically generated user interfaces for Service Agents who want to register to the system as well as for users who want to query events through automated processing of RDF documents. Consequently, with this user interface, registration and querying is done with greater speed and accuracy. Security is also taken into account as services providers are verified by authoritative bodies.

Below we describe the different components of GloServ in detail. Section 2 gives an overview of related service discovery mechanisms. Context-aware service discovery using RDF is discussed in Section 3. The proposed architecture for GloServ is described in detail in Section 4. Finally, our current implementation is discussed in Section 5.

2 Related Work

Service discovery protocols in use today include SLP [13], standardized by the IETF, Sun Microsystem's Jini [16], and Microsoft's UPnP (Universal Plug and Play) [12]. Additionally, [19] describes an Internet telephony gateway location protocol for voice over IP applications. This protocol architecture called Brokered Multicast Advertisments (BMA) addresses the problem of an IP host finding the IP address of the appropriate gateway in order for a call to be made to a user on the PSTN. These service discovery protocols have various similarities and differences. We will look at a few of these protocols briefly in this section.

SLP has three main components: *User Agents (UA)* which perform service discovery on behalf of a client, *Service Agents (SA)* which advertise location and characteristics of the service on behalf of the service, and *Directory Agents (DA)*, which are optional, record available services and also respond to service requests from UAs. In SLP, there are two modes of operation which depend on the existence of the DA. If a DA exists, it records all the information which is advertised by the SAs. UAs then learn of services available by unicasting their requests to the DA. However, in the case where a DA does not exist, UAs repeatedly multicast the same request that they would have unicast to the DAs. SAs listen to these multicast requests and send unicast responses to the UA in case of a service match. The presence of a DA allows for fewer or no multicast messages uses of less bandwidth. The UAs also receive faster responses and thus DAs are usually used in larger networks. Services are advertised and registered using a service URL and a list of service attributes.

Jini is built on top of the Java object and RMI system. Service registries, similar to SLP's DAs, are used to register service proxy objects and act as lookup services. Through a discovery process, a client downloads the service proxy and invokes it to access the service. The Java class hierarchy defines services and their attributes.

UPnP differs from SLP and Jini in that it doesn't have a central service registry but services just multicast their announcements to control points that are listening to these messages. Control points can also multicast discovery messages and search for devices within the system. XML describes the services in greater detail.

SLP and Jini can cover small networks as well as larger enterprise networks whereas UPnP is appropriate for home or small office networks. The query languages for SLP and Java are simple text-based attribute-value pairs. UPnP provides more descriptive queries through XML. The main drawback to these systems is that they do not cover a wide area network that spans the whole Internet, but rather they are limited to a certain vicinity such as home or enterprise networks. SLP and Jini also provide simpler querying mechanism which do not give enough flexibility to the system. GloServ addresses these issues by functioning in both the wide area as well as detailed querying. For example, users can search for museums that exhibit Vermeer paintings in New York City.

3 Context-aware Service Discovery using RDF and RQL

Currently, in service discovery systems, attributes are represented as simple name-value pairs. This limits the ability of the user to search according to its current context by issuing a descriptive query. With the use of RDF, services can be categorized hierarchically where the attributes are described in detail and defined flexibly with RDF schemas.

3.1 RDF

RDF, developed by the World Wide Web Consortium (W3C), is a language for representing information about resources in the World Wide Web. It is based on URI [7] and XML [8] technologies. It was designed to represent metadata about Web resources, such as title or author of a Web page. However, the concept of a "Web resource" has been generalized to include any type of information that can be identified on the Web. RDF is used in situations where information about Web resources needs to be processed by applications. Because of its uniformity, applications can deploy RDF parsers and processing tools in order to exchange this information between applications.

RDF is based on the idea of identifying things using URIs and describing resources in terms of simple properties and property values. RDF is represented in triplets: (subject predicate object). Figure 1 drawn from [15] describes an RDF code and its corresponding graph. One triplet is: *(<http://www.www.w3.org/TR/rdf-syntax-grammar> <http://purl.org/dc/elements/1.1/title> "RDF/XML Syntax Specification(Revised)").* The other triplet illustrates a graph that has no URIref and is a blank node which constructs a hierarchy of different levels. A blank node represents something that does not have a URIref, but can be described in terms of other information. In this case, the blank node represents a person, the editor of the document, and the person is described by his name and home page. The semantics of the following graph is as follows, "the document 'http://www.www.w3.org/TR/rdf-syntax-grammar' has a title 'RDF/XML Syntax Specification (Revised)' and has an editor, the editor has a name 'Dave Beckett' and a home page 'http://purl.org/net/dajobe/' ". Thus, RDF represents the semantics of different URIs on the Web in a flexible yet descriptive manner.

3.2 RDF Schema

The RDF schema [9] is RDF's vocabulary description language that provides a type system for RDF. This is similar in some ways to type systems of object-oriented programming languages such as Java. For example, as in Java, RDF schema allow resources to be defined as instances of one or more classes which contain properties. Properties can define the domain and range of classes. The domain of a property is the class that the property belongs to and the range is the type of values the property can take on.

The mapping of the RDF schema to the RDF document is as follows: Classes are mapped to the "subject" and Properties to the "predicate". Although this is not mandatory, it helps to understand the concept of how RDF schemas

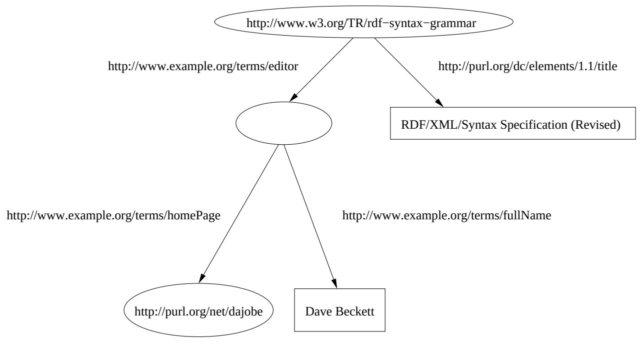
http://www.w3.org/TR/rdf−syntax−grammar
http://www.example.org/terms/editor
http://purl.org/dc/elements/1.1/title
RDF/XML/Syntax Specification (Revised)
http://www.example.org/terms/homePage
http://www.example.org/terms/fullName
http://purl.org/net/dajobe
Dave Beckett

Figure 1. Example of RDF message and corresponding graph

work with RDF documents. Figure 2 shows an example of how an RDF schema relates to an RDF document. The class “Restaurant” has properties “Rating” and “Cuisine” and in the RDF document, “Restaurant” is the subject while “Cuisine” and “Rating” are the predicates. The properties “Cuisine” and “Rating” have their domains set to the class “Restaurant” while their range is of type “Literal”. This means that these properties can be applied to the class “Restaurant” and their values will be of type “Literal”. A property can have more than one domain defined within it which makes the class-property relationship many to many. In this respect, RDF classes and properties differ from programming language types.

3.3 RQL

Various query languages have been developed to query RDF. These are similar to a database query and extract relevant information of a particular RDF message. Two such languages are RDF Query Language (RQL) [21] and RDF Data Query Language (RDQL)) [22]. RQL is a typed functional language, whereas RDQL is an implementation of an SQL-like query language for RDF.

The main feature of RQL that sets it apart from other RDF query languages is that it has richer capabilities such as allowing schemas to be discovered. It can find the description of resources such as the classes under which the resource URL is classified. This gives greater automation for semantic processing and querying. If a user wants to know what class of services a certain URL belongs to, GloServ can present it to the user with a specific query in RQL. It can also perform direct querying of triples of RDF where given either two of the subject, predicate or object values, the third one is returned

Figure 3 shows how a basic RQL query is formed. The

```
 <?xml version="1.0" ?>
 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22−rdf−syntax−ns#"
       xmlns:rdfs="http://www.w3.org/2000/01/rdf−schema#">
   <rdfs:Class rdf:ID="Restaurant">
       <rdfs:comment>"A dining establishment"</rdfs:comment>
       <rdfs:label>Restaurant</rdfs:label>
   </rdfs:Class>
   <rdf:Property rdf:ID="Rating">
     <rdfs:domain rdf:resource="#Restaurant"/>
     <rdfs:range rdf:resource="#Literal" />
   </rdf:Property>
   <rdf:Property rdf:ID="Cuisine">
    <rdfs:domain rdf:resource="#Restaurant"/>
    <rdfs:range rdf:resource="#Literal" />
  </rdf:Property>
</rdf:RDF>
```

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22−rdf−syntax−ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf−schema#">
      xmlns:rest="http://gloserv.com/restaurantSchema#">
  <rest:Restaurant rdf:about="http://restaurant.service.ny.ny.us/PatsPizza.html">
        <rest:Cuisine>Italian</rest:Cuisine>
        <rest:Rating>6</rest:Rating>
  </rest:Restaurant>
  <rest:Restaurant rdf:about="http://restaurant.service.ny.ny.us/Chinese.html">
        <rest:Cuisine>Chinese</rest:Cuisine>
  </rest:Restaurant>
</rdf:RDF>
```

Figure 2. Example of RDF Schemas applied to RDF document

first query is searching for the rating of the PatsPizza restaurant. RQL queries can go down many levels of the hierarchy. In our example, we only show one level. Further examples of querying Classes and Properties are seen in the second and third queires. The second query, determines which classes appear as the domain and range of property “Rating”. The results show that the domain is “Restaurant” and the range is “Literal”. The third query finds all the properties that are defined on class “Restaurant”. That query returns “Rating” and “Cuisine”. Class and Property variables are represented with $ and @ signs respectively.

The Lightweight Directory Access Protocol (LDAP) [24], which is used to search directories, may be used for structuring services and querying for them. The main difference between LDAP and RDF/RQL is that RQL gives the ability to query schemas whereas in LDAP this is not possible. Also, the tree structure is not mandatory in RDF whereas in LDAP it is.

It can be seen from the examples provided that RQL provides a rich set of querying techniques. There are many other different forms of querying in RQL that are beyond the scope of this paper.

```
select Y
from {X}rating{Y}
where X = "http://restaurant.service.ny.ny.us/PatsPizza"
=>Result Y=6
```

```
select $X, $Y from {$X}Rating{$Y}
            OR
select X, Y from Class{X}, Class{Y}, {;X}Rating{;Y}
=> Result: $X= Restaurant, $Y = Literal
```

```
select @P from {;Restaurant}@P
            OR
select P
from Property{P}
where domain(P) = Restaurant
=>Result: Rating, Cuisine
```

Figure 3. Example of RQL queries

4 GloServ Architecture

4.1 Service Hierarchy

The GloServ architecture is similar to DNS in that it contains root name servers and authoritative name servers that manage the information of services. We envision that a separate classification system similar to North American Industry Classification System (NAICS) [1] classifies the hierarchy of services and establishes RDF schemas that describe each type of service. An authority such as ICANN [2] delegates the top level services and events to individual root and authoritative Name Servers. The service categorization is similar to yellow pages directory. Although the categories may change from time to time, they are not expected to change drastically or frequently, which gives us the ability to pre-define the service hierarchy and implement caching in local user agents.

Possible high-level categories for name servers are: events, services, people, or places. The hierarchy goes down all the way to the leaves, which represent the Service Agents. Service Agents (SAs) provide the particular services by registering with GloServ and being inserted into a particular point in the hierarchy as described in the next section.

We have chosen to use URNs (Uniform Resource Names) to reference services in GloServ. One reason for this is that services registered in GloServ may support different types of protocols such as HTTP [11], SIP [20] or others. Therefore, it makes sense to label a service with a name and let each service itself handle the type of protocol mechanism. There can be a general resolution service that maps each service to a list of protocols. The mapping of protocols to services has not been fully researched as of yet.

The URN of a name server is formed according to the hierarchy of the service. We assume that GloServ constructs a hierarchy of services either by location, category, or both. Location-based services have their civil geographical location within the URN to indicate where the particular service is taking place. However, there may very well be services or events that are virtual and not based on a particular location such as in calendar events, which are based on time. We assume both type of hierarchies can exist due to administration and management issues. It is highly unlikely that the maintenance and breakdown of location-based services in the United States, such as restaurants or concerts, will be the same as that in Italy. To have a generic URN such as *urn:gloserv:concert.event* for concert events which is managed by one type of global organization, will be too complicated and hence unrealistic in a real-world scenario. Therefore, we assume that both types of hierarchies can exist and we leave the administration aspect to the various organizations within that country.

The hierarchy of location-based services can be constructed either by specifying the service first or the location. For example, classical music concert servers located in New York City can be specified either by: *urn:gloserv:classical.concert.event.new_york.ny.us* or by *urn:gloserv:new_york.ny.us.classical.concert.event*. The gloserv URN Namespace Identifier (NID) is registered with IANA and specifies global services. The remaining Namespace Specific String (NSS) demonstrates the hierarchy of the service where the top-level name server is either .event or .us. Figure 4 illustrates the hierarchy of GloServ.

The hierarchy of the services is defined with RDF schemas as we discussed in Section 3 for every level within the hierarchy. Every name server within the hierarchy will have its own RDF schema store which describes all the services of its children. Name servers also manage generic registration where new Service Agents can register to and be added to GloServ. GloServ differs from DNS in this respect because instead of having zone files, it uses RDF which not only maps a service to a URN, but also describes each service's properties in greater detail. The addition of services and how RDF stores are managed is described in greater detail below.

We have adopted a similar scheme to DNS in constructing our architecture because it provides scalability. Descriptive URNs of name servers, allow User Agents to perform service querying with greater speed and efficiency. The UAs will be aware of the different levels of the hierarchy

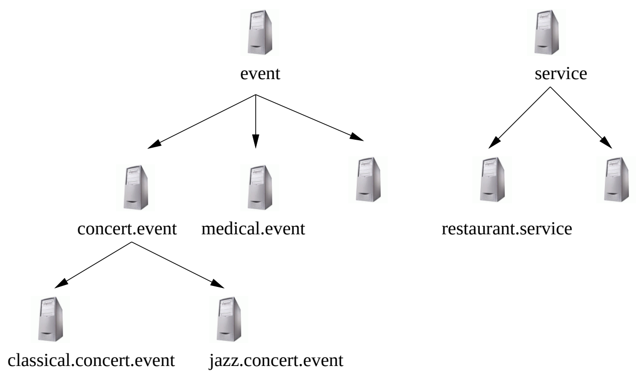

Figure 4. Example of a GloServ hierarchy of Name Servers

of and query directly for a service.

4.2 Service Registration

GloServ has a Services Registry that records all the services in an RDF schema hierarchically and multiple Registrars that handle registration for different service types. There are various options for running registrars in GloServ. There may be a single authoritative registrar for one type of service such as "restaurant" or multiple competing registrars. Registration can either be free, require a fee, by membership in some organization, or by a government license. Free registrations will probably be issued in smaller networks where the registrar is controlled by a network administrator and all services are pretty much trustworthy. On the global scale, however, registration can be more complex. Registrars will most probably require a fee for commercial service providers. This makes sense as service providers will have no incentive to lie about the type of service they provide if they are paying for the service. There may also be multiple registrars for a particular service where registrars compete with one another to store services information and provide different rates, similar to how Internet Service Providers work today. Certain service providers may need government licenses such as doctors, emergency services or plumbers. Service providers will also be verified by Verification Servers, which take in digital signatures of potential registrants and verify that they are legitimate service providers. A Verification Server will have a list of certified digital signatures for every service type. If the registrant's digital signature hashes to the digital signature of its proposed service, then it will be verified, otherwise it will be rejected.

A similar protocol to IRIS (The Internet Registry Information Service) [18] can be adopted for registrars. IRIS is an application layer client-server protocol that establishes a framework for representing the query and result operations of the information services of Internet registries. It specifies this information using XML schemas. Since GloServ will have many different type of Registrars and possibly different methods of registrations, incorporating a protocol similar to IRIS where every registrar can specify its registration mechanism using a schema will be helpful. This is currently an area of research we are looking into.

We have chosen a web interface for service registration. However, we have separated our architecture so that the user interface can be independent of the backend architecture. The registration web form consists of three main parts. The first part displays the hierarchy of the services. The SA administrator checks the place within the hierarchy that it wants to register in. The second part is the security section where the SA enters its digital signature that verifies that it really does provide the service it claims to. Digital signatures are given out by the Verifying Authority Servers that authenticate service providers. These servers may be handled by governmental or private administrative bodies. The registration server verifies the SA by first authenticating it with a verifying authority server. The third part of the form is where the SA's RDF data is entered. Once the registrar verifies the SA, it sends the RDF data to the name server that manages that type of service. The name server stores the RDF data of that SA in its database.

For example, a classical concert event Service Agent in New York City would register with GloServ by specifying that it wants to be inserted under the *urn:gloserv:classical.concert.event.new_york.ny.us* authoritative name server. The *urn:gloserv:classical.concert.event.new_york.ny.us* name server adds the RDF data to its RDF store. The Service Agent will have its own RDF store describing all the services that it provides.

4.3 Querying with local User Agents

This brings us to the querying portion of GloServ. Queries are issued by local User Agents (UAs). These UAs hold a copy of the hierarchy of Service Agents in their RDF store similar to the registration servers. This information is cached and periodically renewed to update any changes within the hierarchy. When a user issues a query, it is automatically connected to a local user agent. The local UA, like the registrar, has a cached copy of the hierarchy of services in an RDF store, which it periodically renews. The main function of the local UA is to query for services on behalf of the user. When a user enters into a room equipped with the GloServ system, it is presented with a form which displays all the categories of services offered. Information about the services is generated from the cached RDF data. Once the user chooses what type of service it wants, the UA connects to that name server and queries for the service. The UA finds the name server similar to the way the SA finds it. It can either directly query for it in the Services Registry or generate a URN and try out first the generic URN for services not based on location and then the concatenated URN of a service and the current location. If a user wants to query for restaurants in New York City, the UA can obtain the registration information of restaurants from the Services Registry and realize that it is a location-based service. However, it can also choose to issue a query to both: *urn:gloserv:restaurant.service* and *urn:gloserv:restaurant.service.new_york.ny.us*. One of these servers will respond and the UA establishes contact with the correct name server. The UA then obtains from the name server RDF metadata that describes the various parameters that the user has to fill out in order to form an RQL query to search the database of that particular SA. This metadata is generated into a form. The metadata is a set of RDF files that describes the user interface of the various properties that can be queried. This is illustrated in the scenario where a user is searching for Italian restaurants in New York City. The local UA connects to the *urn:gloserv:restaurant.registrar.service.new_york.ny.us* server and obtains metadata, generates the GUI and formulates the query for the particular restaurant. The user enters the information it wants to query for by filling out the form. The local UA formulates an RQL query automatically and sends it to the *gloserv:restaurant.service.new_york.ny.us* server. When a user queries for events, the local UA can also subscribe the user to the particular event in which case the user will then receive notifications from the event agent. However, for the purpose of this paper, the event subscription is not discussed.

4.3.1 Context-aware Querying

There may be additional functionality added on to devices where a device may recognize the GloServ system and automatically issue a query pertaining to a user's preference. For instance, a traveler entering into an airport is interested in different touristic events taking place in that city. He may want to have his preferences stored in the device he is using so that the an automatic query can be issued without the user continually entering information about his preferences. The device will first have to recognize that the user is in an airport setting and once it knows that the context of the user is "travel" it will issue the queries that pertain to his travel preferences. Different devices, such as, IR/RF badges or Pocket PCs, can be used to store user information and to query for services.

This functionality can either be part of GloServ or it can be integrated within another context-aware system. Since GloServ is a service discovery architecture, it is better to separate this functionality and let another system handle the automated issuing of queries. [6] describes a ubiquitous computing system using SIP. It builds on CINEMA (Columbia InterNet Extensible Multimedia Architecture) [23] infrastructure and uses SLP for service discovery. This system can be integrated with GloServ where instead of issuing SLP queries, it can issue GloServ queries and provide the user with a context-aware environment.

5 Implementation and Future Work

We are currently implementing a prototype version of GloServ. We have chosen to use Sesame [10] for creating and storing RDF records. Sesame is an experimental RDF store which implements different protocols in its queries, such as HTTP, Java RMI and SOAP. Other protocol handlers can also be added quite easily. Sesame also provides a Java API for RDF parsing and database management. It also provides a web interface to manually update RDF stores by adding, deleting or editing RDF entries.

In our previous work, we created a generic event notification system using the Session Initiation Protocol (SIP) and XML messages [4]. XML schemas that describe the parameters of an event were processed to automatically generate graphical user interfaces for the subscriber to input his information.

We are adopting a similar method in our implementation for GloServ. We use the RDF schemas to generate the user interface for registration and querying. Currently, we are implementing the registration functionality of GloServ. We are also looking into using OWL [3] for automated registration and service discovery. OWL and RDF are much of the same thing, but OWL is a stronger language with greater machine interpretability than RDF. OWL also comes with a larger vocabulary and stronger syntax than RDF which may be useful for describing services within a hierarchy.

6 Conclusion

We have described GloServ, a global service discovery architecture that functions both on an wide area as well as a local area network. GloServ applies to a broad range of services since services are defined flexibly with the use of RDF schemas. It also administers automatic and sophisticated querying using RDF and RQL. Querying can either be done manually or automatically using sensor technology which results in a seamless discovery of services.

7 Acknowledgement

This work is supported by a grant from Nokia Research. We would also like to acknowledge the contributions of

Dirk Trossen and Dana Pavel from Nokia Research.

References

[1] North American Industry Classification System http://www.naics.com.

[2] Internet Corporation for Assigned Names and Numbers http://www.icann.org.

[3] OWL http://www.w3.org/2004/OWL/.

[4] K. Arabshian and H. Schulzrinne. A generic event notification system using XML and SIP. In *New York Metro Area Networking Workshop 2003*, Sept. 2003.

[5] D. J. Beckett and B. McBride. RDF/XML syntax specification (revised). W3c proposed recommendation, World Wide Web Consortium, Dec. 2003.

[6] S. Berger, H. Schulzrinne, S. Sidiroglou, and X. Wu. Ubiquitous computing using SIP. In *ACM NOSSDAV 2003*, June 2003.

[7] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifiers (URI): generic syntax. RFC 2396, Internet Engineering Task Force, Aug. 1998.

[8] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible markup language (XML) 1.0 (second edition). Technical report, Oct. 2000.

[9] D. Brickly and R. Guha. Rdf vocabulary description language 1.0: Rdf schema. W3c proposed recommendation, World Wide Web Consortium, Feb. 2004.

[10] J. Broekstra and A. Kampman. Sesame: A generic architecture for storing and querying RDF and RDF schema. Technical report, Aidministrator Nederland, Oct. 2001.

[11] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. J. Leach, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1. RFC 2616, Internet Engineering Task Force, June 1999.

[12] U. Forum. UPnP device architecture 1.0. Technical report, Dec. 2003.

[13] E. Guttman, C. E. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. RFC 2608, Internet Engineering Task Force, June 1999.

[14] O. Lassila and R. R. Swick. Resource description framework (RDF) model and syntax specification. W3c recommendation, World Wide Web Consortium, Feb. 1999.

[15] F. Manola, E. Miller, and B. McBride. RDF primer. W3c proposed recommendation, World Wide Web Consortium, Dec. 2003.

[16] S. Microsystems. Jini architectural overview. Technical report, 1999.

[17] P. V. Mockapetris. Domain names: Concepts and facilities. RFC 882, Internet Engineering Task Force, Nov. 1983.

[18] A. Newton. IRIS - the Internet registry information service (IRIS) core protocol. Internet Draft draft-ietf-crisp-iris-core-04, Internet Engineering Task Force, Oct. 2003. Work in progress.

[19] J. Rosenberg and H. Schulzrinne. Internet telephony gateway location. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, pages 488–496, San Francisco, California, March/April 1998.

[20] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. R. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: session initiation protocol. RFC 3261, Internet Engineering Task Force, June 2002.

[21] M. Scholl, editor. *RQL: A Declarative Query Language for RDF*. WWW, May 2002.

[22] A. Seaborne. JENA tutorial a programmer's introduction to RDQL. Apr. 2002.

[23] K. Singh, W. Jiang, J. Lennox, S. Narayanan, and H. Schulzrinne. CINEMA: columbia internet extensible multimedia architecture. technical report CUCS-011-02, Department of Computer Science, Columbia University, New York, New York, May 2002.

[24] M. Wahl, T. Howes, and S. E. Kille. Lightweight directory access protocol (v3). RFC 2251, Internet Engineering Task Force, Dec. 1997.